CONFIGURATION MANAGEMENT APPARATUS AND RELATED METHODS

FIELD OF THE INVENTION

[0001] The present invention relates to management and maintenance systems and, more particularly, to an apparatus for managing configuration of an element such as a mobile platform.

BACKGROUND OF THE INVENTION

[0002] Maintaining a fleet of aircraft involves stringent testing and maintenance of hardware and software components of each aircraft. Records of the aircraft components are kept so that maintenance mechanics may determine what versions of hardware and/or software are in use on a given aircraft at a particular time. Such recordkeeping typically has been performed by the mechanics, who record aircraft configuration information on paper when maintenance is performed.

SUMMARY OF THE INVENTION

[0003] The present invention, in one preferred implementation, is directed to an apparatus for managing configuration of an element. The apparatus includes a computer that connects with the element to receive data from the element, and a server in the computer that includes the data in one or more web pages.

[0004] In another configuration, an apparatus for performing configuration management relative to an element includes a computer and a configuration file that receives data from the element. The configuration file is accessible by the computer. The computer formats the data from the configuration file for presentation as one or more web pages. The formatting is performed standalone using a server module of the computer.

[0005] In another implementation, a method of managing configuration of an element includes connecting a computer to the element. Data is caused to be transferred from the element to the computer. The transferred data is viewed in one or more web pages formatted by the computer standalone.

[0006] In yet another implementation, a computer-implemented method of providing configuration management relative to an element includes connecting with and receiving data from the element, and formatting the data in one or more web pages. The formatting is performed standalone.

[0007] The features, functions, and advantages can be achieved independently in various embodiments of the present inventions or may be combined in yet other embodiments.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] The present invention will become more fully understood from the detailed description and the accompanying drawings, wherein:

[0009] Figure 1 is a block diagram view of an apparatus for managing an element in accordance with one configuration of the present invention;

[0010] Figure 2 is a block diagram of software modules of a computer in accordance with one configuration of the present invention;

[0011] Figure 3 is a view of a web page configured in accordance with one implementation of the present invention;

[0012] Figure 4 is a view of a web page configured in accordance with one implementation of the present invention;

[0013] Figure 5 is a view of a web page configured in accordance with one implementation of the present invention;

[0014] Figure 6 is a view of a web page configured in accordance with one implementation of the present invention;

[0015] Figure 7 is a listing of "htm" code configured in accordance with one implementation of the present invention;

[0016] Figure 8 is a listing of "htm" code configured in accordance with one implementation of the present invention;

[0017] Figure 9A is a listing of a portion of a configuration file in accordance with one implementation of the present invention;

[0018] Figure 9B is a listing of a portion of a configuration file in accordance with one implementation of the present invention;

[0019] Figure 10 is a listing of HTML code configured in accordance with one implementation of the present invention;

[0020] Figure 11A is a listing of HTML code configured in accordance with one implementation of the present invention; and

[0021] Figure 11B is a listing of HTML code configured in accordance with one implementation of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0022] The following description of the preferred systems and methods is merely exemplary in nature and is in no way intended to limit the invention, its application, or uses. Although configurations of the present invention are described herein with reference to a fleet of aircraft, configurations of the invention are not so limited. Configurations are contemplated in connection with various systems and networks, including but not limited to other mobile platforms such as ships and trains. Various configurations could be implemented in connection with many different types of networks, systems and/or elements for which it is desired to provide management and/or maintenance.

[0023] An apparatus for managing an element in accordance with one configuration of the present invention is indicated generally in Figure 1 by reference number 10. A plurality of elements 28 are included in a system 20. The elements 28 may be, for example, aircraft in a fleet. A management node 34, *e.g.,* a network operations center, includes one or more processors 40 used, for example, to track and/or analyze data collected from the element(s) 28 as further described below. Configurations of the present invention also are contemplated in connection with systems which do not include a management node. The invention can also be implemented in connection with an element that is not part of a system.

[0024] The apparatus 10 includes a computer 56, for example, a laptop computer having a display screen 62 and a keyboard or other user input device 68. The computer 56 has a processor 72 and memory 76. As further described below, the computer 56 is operable to connect with an aircraft 28 to transmit data to and/or receive data from the aircraft 28. The computer 56 also is operable to include data received from the aircraft 28 in one or more web pages and to present such web pages, *e.g.,* display such web page(s) on the display screen 62. The computer 56 can be used standalone to format and/or display such web

pages. Unless otherwise stated, the computer 56 may be "connected" with an element 28 via wired or wireless connection.

[0025] When, for example, it is desired to perform aircraft management relative to one of the aircraft 28, a user connects the computer 56 to the aircraft and causes data to be transferred from the aircraft 28 to the computer 56. Such data may be displayed on the screen 62 in one or more web pages formatted by the computer standalone. Based, for example, on the transferred data, the user may use one of the web page(s) to select a management function as further described below.

[0026] The computer 56 includes a plurality of software and data modules as shown in Figure 2. A web browser module 100 provides an interface between the computer 56 and a user of the computer 56. The computer 56 also includes a web server module 104 further described below. The computer 56 may retrieve data, for example, from one or more line replaceable modules (LRMs) 112 of the aircraft 28. The computer 56 may use simple network management protocol (SNMP) to retrieve such data, although other protocols could be used. Data retrieved from the aircraft 28 is stored in a configuration file 116 of the computer 56. The computer 56 may also be configured to send data to the aircraft 28. For example, one or more software modules 120 may be sent to update one or more LRMs 112. In the present configuration, the computer 56 includes an optional translation file 128 further described below.

[0027] The server 104 uses one or more web page descriptions or markups 134 written, for example, in hypertext markup language (HTML) to output web pages. A given web page description 134 may include static content, *i.e.,* content that does not change over time. Additionally or alternatively, one or more of the descriptions 134 may include one or more constructs 136 used by the server 104 to insert dynamic content (*i.e.,* content that may change over time) into a web page as further described below. Such construct(s) 136 may be used to obtain data from the configuration file 116 and to insert the data and/or content derived from such data into a web page. A construct processing module 140 written, for example, in perl, is executable by the server 104 via a script wrapper 144 (referred to in the present configuration as "cgi_dpu") to process the construct(s) 136 as further described below. The script 144 in the present

configuration also is written in perl. Other languages could be used, however, for the module 140 and script 144. It also should be noted that other languages besides HTML could be used to output web pages in other implementations of the present invention. The term "web page" as used herein refers to a page formatted using the server module 104. Although the invention can be implemented without accessing the Internet, pages formatted using the server 104 could be transported over the Internet in other implementations.

[0028] In one configuration, the construct processing module 140 uses a common gateway interface (CGI) of the server 104 to obtain data from the configuration file 116. The construct processing module 140, referred to in the present configuration as "dpu.pm", may have a size, for example, of approximately 30 kilobytes. The module 140 is placed, for example, in a cgi-bin directory of the server 104.

[0029] For a typical static link to a HTML page, a web page description may include the following:

Go to Main

[0030] For a link with dynamic content update, a web page description may instead include the following:

Go to Main

[0031] The foregoing statement instructs the server 104 to execute the script "cgi_dpu" and pass to it the name of the web page, in the present example, "main.htm". The optional notation ".htm" (or other notation) may be used instead of the extension ".html" to indicate a page that includes one or more constructs for dynamic update as further described below.

[0032] In one configuration, one or more data replacement constructs, *e.g.,* as described in Table 1, may be used in creating web pages.

Table 1. Data Replacement Constructs [[ ]]
[[param_name]] - Look up param_name from configuration file and replace [[ ]] with the param_value from the configuration file. [[ ]] can be nested and is processed from right to left with the innermost [[ ]] evaluated first.
[[eval (..)]] - Evaluate expression inside ( ) using perl and replace [[ ]] with the return result.
[[system (..)]] - Execute system command inside ( ) and capture a standard system output (stdout) and replace [[ ]] with the captured result.

[0033] Additionally or alternatively, one or more logic processing constructs may be used as shown in Table 2.

Table 2. Logic Processing Constructs [% %]

[%if (..)%] - [%endif%]

Allow inclusion of a block of htm code if ( ) is true. Also supports [%elseif (..)%] and [%else%] keywords within the "if" and "endif" block. Nested 'if' block is acceptable.

[%foreach (..)%] - [%endfor%]

Allow looping of a block of html code with a specific loop value.

Following are three exemplary constructs for "foreach":

1.

[%foreach ("xx", yy", "zz")%]

htm code

[%endfor%]

2.

[%foreach (n..m)%]

htm code

[%endfor%]

3.

[%foreach (@filename)%]

htm code

[%endfor%]

Note: n and m are integer values and n is less than or equal to m.

"filename" is the name of a file that contains possible loop values, one on each line.

Also supports [%continue%] and [%break%] within a "foreach" block.

Nested "foreach" block is acceptable.

[%set (..)%] - Set a variable to a particular value.

[%eval (..)%] - Allow execution of a perl command.

[%system (..)%] - Allow execution of a system command.

[%include (..)%] - Allow inclusion of another htm file at this location.

[0034] The following examples demonstrate how dynamic content update can be provided. Preferably, the construct processing module 140 processes a given page including a construct 136 after the configuration file 116

has been updated. A data replacement construct may be used in the following manner. A line may appear in a web page description 134 as follows:

<td>[[abc]]<td>

[0035] The construct processing module 140 captures content inside the "[[ ]]", deletes all leading and trailing blanks and uses a resulting string as a name to fetch a value from the configuration file. For example, where a string "abc=snmp connection" is stored in the configuration file 116, the following result is sent to the browser 100:

<td>snmp connection<td>

[0036] As another example, a line may appear as follows:

<td>[[abc]]([[link]]):</td><td>[[[[abc]].[[link]]]]</td>

[0037] Where the configuration file 116 includes strings "link=1" and "snmp connection.1=OK", the following result is sent to the browser 100:

<td>snmp connection(1):</td><td>OK</td>

[0038] The foregoing result is obtained using the following processing sequence, in which processing is performed right-to-left with innermost "[[ ]]" processed first:

<td>[[abc]]([[link]]):</td><td>[[[[abc]].1]]</td>
<td>[[abc]]([[link]]):</td><td>[[snmp connection.1]]</td>
<td>[[abc]]([[link]]):</td><td>OK</td>
<td>[[abc]](1):</td><td>OK</td>
<td>snmp connection(1):</td><td>OK</td>

[0039] As another example, a line may appear as follows:

<td>[[eval ([[link]]+1)]]</td>

[0040] Processing the line results in the following string:

<td>2</td>

[0041] As another example, a line may appear as follows:

<td>The time is: [[system (date)]]</td>

[0042] A result of execution of a system command 'date' is inserted to produce the following:

<td>The time is: Wed Mar 10 09:45:22 PST 2004</td>

[0043] The following example shows how a logic processing construct may be used. A foreach block may be provided as follows:

[%foreach (1,2,3)%]
<tr><td>[[abc]]([[foreach_var]]):</td><td>[[[[abc]].[[foreach_var]]]]</td></tr>
[%endfor%]

[0044] The foregoing lines provide looping of HTML code within the foreach block three times with loop values of 1, 2 and 3. A predefined variable 'foreach_var' can be used to capture the loop value for each loop. It is assumed that the configuration file 116 includes the following values:

snmp connection.1=OK
snmp connection.2=NOT OK
snmp connection.3=OK

[0045] The following HTML code is produced:

<tr><td>snmp connection(1):</td><td>OK</td></tr>
<tr><td>snmp connection(2):</td><td>NOT OK</td></tr>
<tr><td>snmp connection(3):</td><td>OK</td></tr>

[0046] The following code illustrates how "[%if..." may be used:

[%if ("[[abc]].[[link]]" eq "OK")%]
<tr><td>[[abc]]([[link]]):</td><td>[[[[abc]].[[link]]]]</td></tr>
[%elseif ("[[abc]].[[link]]" ne "NOT OK")%]
<tr><td>[[abc]]([[link]]):</td><td bgcolor=yellow>[[[[abc]]. [[link]]]]</td></tr>
[%else%]
<tr><td>[[abc]]([[link]]):</td><td bgcolor=red>[[[[abc]]. [[link]]]]</td></tr>
[%endif%]

[0047] Using the previously described configuration file values and if link=1, then the following code is produced:

```
<tr><td>snmp connection(1):</td><td>OK</td></tr>
```

[0048] If link=2, then the following code is produced:

```
<tr><td>snmp connection(2):</td><td bgcolor=red>NOT OK</td></tr>
```

[0049] The "if" keyword can support logical compare operators available in perl, *e.g.,* "==", "!=", ">=" and "<=" (for numeric), "eq", "ne", and "gt" and "lt" (for string).

[0050] The following code illustrates how "[%set ..." may be used:

```
[%set (connected = [[[[abc]].[[link]]]])%]
```

[0051] Using the previously described configuration file values and where link=1, the foregoing construct sets a variable 'connected' within a web page to "OK". To provide access to a "set" variable, the "[[ ]]" construct is used around that variable, for example:

```
[%if ("[[connected]]" eq "OK") %]
    ...
[%endif%]
```

[0052] The following code illustrates how "[%system ...]" may be used:

```
[%system (./updateParameter)%]
```

[0053] The foregoing line causes the perl module 140 to call a system command "updateParameter" residing in a current directory. In one implementation, to ensure that the configuration file 116 has been updated, the same or a similar statement is put near the beginning of a given htm file. Since such a system call is performed synchronously, the perl module 140 waits until the system call is completed before continuing to process the rest of the htm file.

[0054] The following code illustrates how "[%include ..." may be used:

```
[%include(my_header.htm)%]
```

[0055] The foregoing statement causes the perl module 140 to read in a file "my_header.htm" in the current directory and process it as if it were part of the htm file being processed.

[0056] The cgi script wrapper 144, named, for example, ""cgi_dpu", may be configured as follows:

```
#!/usr/bin/perl -w
use dpu;
$cgi_dpu = new dpu("conf_file", "xlate_file");
$cgi_dpu->process();
```

[0057] The foregoing script may be made executable as follows:

```
(chmod +x cgi_dpu)
```

[0058] The foregoing script provides the perl module 140 with names of the configuration file 116 and the translation file 128 further described below. If access to the translation file is not desired, a NULL pointer may be passed to the perl module 140. If the perl module 140 is not installed in /usr/bin, an appropriate path may be entered in the script wrapper 144.

[0059] As previously discussed, the configuration file 116 includes information in the following format:

```
name=data
```

where "name" is a parameter name and can be any ASCII text. One or more spaces can be included within the "name". "Data" is a parameter value and can be any ASCII text. One or more spaces can be included within the "data". In the present implementation, leading or trailing spaces are not used. A pound sign ("#") in the first character of a line designates a comment line and is ignored.

[0060] The translation file 128 is used to define one or more callback functions (*e.g.*, system commands) to perform pre-get and pre-set processing as further described below. When a parameter is not defined in the configuration file 116, an entry can be included in the translation file 128 to provide content for such parameter. The translation file 128 includes information in the following format:

```
name:option=cmd
```

where "name" is a parameter name and can be any ASCII text. One or more spaces can be included within the "name". The "option" can be, *e.g.,* "iget", "iset", "get" and/or "set" as described below. The "cmd" is a system command (which may be, for example, a script or a program) and can be any ASCII text. Leading or trailing spaces are not used in the present implementation.

[0061] The "iget" option is used to define a system call to inform the current application that a parameter is about to be fetched from the configuration file 116. The "iget" option basically is a pre-"get" command option. A system call associated with an "iget" option exits with a value of zero (OK) or non-zero (NOT OK). If the exit value is OK, the perl module 140 will continue to try to fetch the replacement value string from the configuration file. If the exit value is NOT OK, the perl module 140 will skip the configuration file 116 and proceed to perform the "get" option described below.

[0062] The "iget" option may be used, for example, in the following manner:

current_transfer_speed:iget=./calculate_transfer_speed

Before a value "current_transfer_speed" is fetched from the configuration file 116, the foregoing statement may be used to allow a value for current transfer speed to be calculated and stored in the configuration file 116. The newly calculated value may be stored in the configuration file 116 before being fetched for display in a web page.

[0063] The "iset" option is used to define a system call to inform the current application that a value string is about to be stored to the configuration file 116 for a particular parameter. The value string may be entered, for example, by a user via a web page as further described below. The "iset" option basically is a pre-"set" command option. Before the command is executed, the value string is appended to the command line so the command can determine what to do with the new value string. For example, a line of code may appear as follows:

Aircraft ID:iset=./validate 'Aircraft ID'

The foregoing statement causes a system command called "validate" to be executed with two passed arguments. The first argument is 'Aircraft ID', and the second argument is the value string that is appended to the command line as previously described. If the appended string is validated, a zero (OK) exit value is returned. If the string is not validated, a nonzero (NOT OK) exit value is returned. If the exit value is OK, the new value string is stored in the configuration file. If the exit value is NOT OK, the configuration file is not updated and the "set" option is performed.

[0064] The "get" option is used to define a system call to get a value for a parameter, for example, if the parameter does not exist in the configuration file and/or if the "iget" option exits with a NOT OK status as described above. When the "get" command is executed, output is captured from a standard system output (stdout) as a value string for the parameter referred to by "name". For example, the following statement includes the "get" option:

DATE:get=date

To provide a parameter value for "DATE", a system call is made for "date", that is, a command "date" is executed that provides the current date in "stdout". The current date is captured from "stdout" and displayed as the value string for "DATE".

[0065] The "set" option is used to define a system call to set a value for a parameter, for example, if the parameter does not exist in the configuration file and/or if the "iset" option exits with a NOT OK status as described above. Before the command is executed, the value string is appended to the command line so the command can determine what to do with the new value string.

[0066] The "set" option may be used, for example, in the following manner:

PASSWD:set=set_passwd

The foregoing statement allows a user to enter a password that is not stored in the configuration file 116. The password entered by the user is passed as an argument to and is processed by the command "set_passwd", which may save the password in a privileged-access file.

[0067] The following two statements are examples of what might be included in the translation file 128:

DATE_TIME:get=date –u '+%A, %d-%b-%y %T GMT'

The foregoing statement may be used to obtain a current system date and time, which changes constantly and thus is not stored in the configuration file 116.

CLEAR_SHELL_WINDOW:get=rm –f shell_window.txt;touch shell_window.txt

The foregoing statement may be used, for example, to perform initialization prior to web page initialization for display. Two commands are executed to obtain a

value for the parameter "CLEAR_SHELL_WINDOW". The first command, "rm –f shell_window.txt", deletes a file name "shell_window.txt". The second command, "touch shell_window.txt", creates a file name "shell_window.txt" having zero length. Since no value is sent to "stdout", the "get" option captures nothing, and so the value for "CLEAR_SHELL_WINDOW" will have an empty string.

[0068] One or more system commands may be implemented to access the configuration file 116 or the translation file 128 (also referred to respectively, for example, as "conf_file" and "xlate_file"). Several of such commands are shown in Table 3.

Table 3. System commands

configGet filename param_name

Examples:

configGet cbb_server.conf 'Aircraft ID'

The foregoing statement gets the parameter 'Aircraft ID' from a configuration file 'cbb_server.conf' and displays the parameter value to "stdout".

configGet cbb_xlate.conf "UNAME:get'

The foregoing statement gets the parameter 'UNAME:get' from a translation file 'cbb_xlate.conf' and displays the parameter value to "stdout".

configSet filename param_name param_value

Examples:

configSet cbb_server.conf 'Aircraft ID' 'x747-12345'

configSet cbb_xlate.conf "UNAME:get' 'uname -a'

The "configSet" command does not create an entry in a file designated by "filename", and the command is ignored if the entry does not exist.

configAdd filename param_name param_value

Examples:

configAdd cbb_server.conf 'Aircraft ID' 'x747-12345'

configAdd cbb_xlate.conf "UNAME:get' 'uname -a'

The command "configAdd" creates an entry in a file designated by "filename" if the entry does not exist.

configDelete filename param_name

Examples:

configDelete cbb_server.conf 'Aircraft ID'

configDelete cbb_xlate.conf 'Uname:get'

configDisplay filename param_name

The "configDisplay" command compares each line with a supply string and displays all lines containing the supply string.

Examples:

configDisplay cbb_server.conf 'Aircraft'

configDisplay cbb_xlate.conf 'UNAME'

[0069] An example shall now be described wherein a configuration of the foregoing apparatus is used in managing various systems and components of an aircraft. Generally, one or more web pages may be formatted using the server 104 and displayed using the browser 100. Such pages may include dynamic data pertaining to such aircraft. An exemplary web page is indicated generally in Figure 3 by reference number 200. The page 200 includes a top frame area 202 in which, for example, a company logo 204 is displayed. The area 202 also includes aircraft information 208 specific to the given aircraft, including Aircraft Identifier, Tail Number and Aircraft Type. A Computer Software Version 212 also

is displayed to keep track of which software version is being executed by the computer 56.

[0070] A button 218 may be activated by a user to update the aircraft information 208. The button 218 preferably is activated each time the computer 56 is connected to a new aircraft. Activating the button 218 causes the aircraft information fields 208 to be updated with data obtained from the aircraft via the configuration file 116.

[0071] The page 208 also includes a menu 224 of various functions that maybe activated by a user of the computer 56. Such functions generally may include but are not limited to configuration management functions, software upgrade functions, health status functions, troubleshooting functions, and engineering functions. The menu 224 may change dependent on an operating mode (*e.g.,* whether the computer 56 is operating in a normal maintenance mode, or in an engineering mode in which additional diagnostic functions may be made available to the user).

[0072] Generally, web page display areas may be configured to provide one or more functions as selected by the user. For example, as shown in Figure 3, a display area indicated generally by reference number 230 allows the user to perform a software upgrade for a LRM. A source for upgrade software (*e.g.,* CDROM or hard disk) may be designated in an area 234. LRM-specific information may be entered via buttons 240. To access high-level functions, the user may select an access level and enter a password in an area 246.

[0073] Another exemplary page is indicated generally by reference number 300 in Figure 4. A display area indicated generally by reference number 308 allows the user to review, date-identify and save in the computer 56 configuration data pertaining to various LRMs for a given aircraft. In one configuration such data can be retrieved from the computer 56 and delivered to the management node 34, for example, to be included in a management data base.

[0074] Another exemplary page is indicated generally by reference number 400 in Figure 5. A display area indicated generally by reference number 408 shows communication status with respect to a plurality of LRUs. Columns 412, 416 and 420 show respectively LRU name, IP address and ping status. Up

time since last reboot also is shown for each LRU in a column 424. Data displayed in the columns 412, 416, 420 and 424 may be obtained from the configuration file 116, which preferably has been updated with current aircraft data by activation of the area 218.

[0075] Another page according to one configuration is indicated generally by reference number 500 in Figure 6. The page 500 is displayed in response to a user having selected "Load To MSD" from the menu 224. A column 508 describes hardware and a column 516 describes part numbers affected by such a load operation. As previously discussed with reference to Figure 3, the menu 224 may change according to an operating mode of the computer 56. Mode may be determined by a user access level and password entered in the page area 246 (shown in Figure 3). As shown in Figure 6, the menu 224 is extended to include menu items 224a when the computer 56 operates in an "Engineering" mode.

[0076] Other and additional functions may be made available via other and additional web pages. For example, one such web page (not shown) may be used to obtain SNMP data from various LRMs. Other and additional system status information such as aircraft-specific pin configuration and maintenance items (also referred to as discrete status), may be displayed on one or more web pages. Displays of such content may be refreshed periodically, *e.g.,* every ten seconds. In one configuration, the computer 56 includes a Linux operating system and the server 104 is an Apache server having a perl interpreter. Other and additional operating systems and/or servers could be used in other configurations.

[0077] Exemplary "htm" code for the top frame area 202 is indicated generally in Figure 7 by reference number 600. Exemplary "htm" code for the menu area 224 is indicated generally in Figure 8 by reference number 650. In Figures 9A and 9B are shown portions (indicated generally by reference numbers 680a and 680b) of the configuration file 116 used in configuring the page shown in Figure 1. A listing of HTML code is indicated generally in Figure 10 by reference number 700. The code 700, which produces the top frame area 202, results from processing the "htm" code 600 using the "dpu.pm" module 140 and the configuration file 116 as previously described. Portions of HTML code are

indicated generally in Figures 11A and 11B by reference numbers 750a and 750b. The code 750, which produces the menu frame area 224, results from processing the "htm" code 650 using the "dpu.pm" module 140 and the configuration file 116 as previously described.

[0078] The foregoing apparatus and methods make it convenient to fetch, store and display information from an aircraft or other system element. The apparatus is useful for upgrading software for a LRM, gathering fault and performance characteristics, and performing troubleshooting using various troubleshooting tools. Additionally, histories of information pertinent to various aircraft in a fleet could be compiled using configurations of the present invention. In systems in which a central management facility collects and uses such information, the foregoing apparatus and methods can simplify the collection of such information.

[0079] Because web server and internet browser software are used as an engine and user interface, an easily understandable language such as HTML can be used to format data for display. Even someone who is not technically knowledgeable could format data in such manner. The above-described constructs are flexible yet simple, and make it unnecessary to install a bulky, complicated server-side scripting package. The foregoing server performs as a standalone engine that provides a simple way to update web pages with dynamic content.

[0080] While various preferred embodiments have been described, those skilled in the art will recognize modifications or variations which might be made without departing from the inventive concept. The examples illustrate the invention and are not intended to limit it. Therefore, the description and claims should be interpreted liberally with only such limitation as is necessary in view of the pertinent prior art.